Продолжаем совершенствовать класс DArray и переопределим для него операции присваивания:

+=, -=, *=, /=, %= и т.п.

Очевидно, их можно было бы использовать в следующей логике работы объектов класса DArray:

```language-cpp
#include <iostream>
#include "darray.h"

int main()
{
    DArray ar1, ar2;

    ar1[10] = 100;
    ar1[0] += 2;
    ar1[2] = 7; ar1[2] *= 3;
    ar1[3] = 11; ar1[3] %= 4;
    double v = ar1[14];

    std::cout << v << std::endl;

    for(int i = 0;i < ar1.size(); ++i)
        std::cout << ar1.get_data()[i] << " ";

    return 0;
}
```

Давайте пропишем эти операции. Так как они применяются при индексировании к отдельным элементам динамического массива, то их следует располагать в классе Item:

```language-cpp
    class Item {
        DArray* current {nullptr};
        int index {-1};

    public:
        Item(DArray* obj, int idx) : current(obj), index(idx)
            { }

        operator int() const;
        int operator=(int right);
```

```
        int operator+=(int right);
        int operator*=(int right);
        int operator%=(int right);
    };
```

Я прописал только три расширенные операции присваивания, но вы самостоятельно легко можете добавить все остальные. Соответственно, реализации этих методов будут располагаться в файле darray.cpp:

```language-cpp
int DArray::Item::operator+=(int right)
{
    if(index >= current->length || index < 0)
        return right; // операцию += можно выполнять только с существующими
элементами массива

    current->data[index] += right;
    return current->data[index];
}


int DArray::Item::operator*=(int right)
{
    if(index >= current->length || index < 0)
        return right; // операцию *= можно выполнять только с существующими
элементами массива

    current->data[index] *= right;
    return current->data[index];
}

int DArray::Item::operator%=(int right)
{
    if(index >= current->length || index < 0)
        return right; // операцию %= можно выполнять только с существующими
элементами массива

    current->data[index] %= right;
    return current->data[index];
}
```

Мы здесь, фактически, получаем дублирование кода. Как можно было бы это поправить? Один из вариантов – вынести общие действия в отдельный метод. Например, так:

```language-cpp
    class Item {
        enum type_assign {
            iadd_operator, imul_operator, iddiv_operator
        };

        DArray* current {nullptr};
        int index {-1};

        int _assign_operator(int right, type_assign t);
    public:
    ...
    };
```

Со следующей реализацией:

```language-cpp
int DArray::Item::_assign_operator(int right, type_assign t)
{
    if(index >= current->length || index < 0)
        return right; // операции +=, -=, *=, /= и т.п. можно выполнять только с записанными элементами массива

    switch(t) {
        case iadd_operator:
            current->data[index] += right;
            break;
        case imul_operator:
            current->data[index] *= right;
            break;
        case iddiv_operator:
            current->data[index] %= right;
            break;
    }
    return current->data[index];
}
```

А в методах операций лишь вызывать этот приватный метод:

```language-cpp
int DArray::Item::operator+=(int right)
{
    return _assign_operator(right, iadd_operator);
}


int DArray::Item::operator*=(int right)
{
    return _assign_operator(right, imul_operator);
```

```
}

int DArray::Item::operator%=(int right)
{
    return _assign_operator(right, iddiv_operator);
}
```

Получим тот же самый результат.

Конечно, все расширенные операции присваивания можно переопределять только на уровне методов класса, но не функций, так же, как и операцию простого присваивания.

В заключение этого занятия продемонстрирую пример переопределения операции += на уровне объектов класса DArray.

Операция += в нашем примере будет добавлять элементы одного массива в конец другого:

```language-cpp
ar1 += ar2;
```

Метод этой операции, очевидно, нужно записать в классе DArray. Объявим его прототип следующим образом:

```language-cpp
const DArray& operator+=(const DArray& other);
```

Обратите внимание, здесь параметр передается по константной ссылке и возвращается тоже константная ссылка. Почему именно так? Да, для того, чтобы не выполнять лишних операций копирования объектов динамического массива. Как вы понимаете, это может заметно тормозить выполнение программы, тогда как избежать этого очень просто за счет применения ссылок.

Также обратите внимание, что операция += не обязана возвращать константную ссылку на объект класса DArray. Как вариант, можно записать тип void. Но по философии языка C++ операция += возвращает сформированный результат. Поэтому я повторяю эту идею.

Давайте теперь пропишем реализацию этого метода в файле darray.cpp:

```language-cpp
const DArray& DArray::operator+=(const DArray& other)
{
    int size_new = length + other.length;
    if(size_new > max_length_array)
```

```
        size_new = max_length_array;

    _resize_array(size_new);

    for(int i = length, j = 0; i < size_new; ++i, ++j)
        data[i] = other.data[j];

    length = size_new;
    return *this;
}
```

Здесь все достаточно очевидно. Увеличивается (при необходимости) физический размер массива data. И, затем, добавляются в него новые данные из переданного массива. В конце возвращаем результат в виде текущего объекта.

Теперь, мы можем выполнять операции соединения двух динамических массивов операцией +=. По аналогии реализуйте операцию сложения массивов, которая так же соединяет два массива во временный объект класса DArray. Сами объекты, участвующие в сложении, менять свое состояние не должны.